IN THE UNITED STATES PATENT AND TRADEMARK OFFICE
BEFORE THE BOARD OF PATENT APPEALS AND INTERFERENCES

Application No.: 10/602,551	§	Examiner: Dao, Thuy Chan
Filed: June 24, 2003	§	Group/Art Unit: 2192
Inventor(s):	§	Atty. Dkt. No: 5150-80201
Thomas A. Makowski, Rajesh	§	
Vaidya, Deborah E. Bryant and	§	
Brian M. Johnson	§	
Title: TASK BASED	§	
POLYMORPHIC	§	
GRAPHICAL PROGRAM	§	
FUNCTION NODES	§	
	§	

REPLY BRIEF

Box: Appeal Brief - Patents
Commissioner for Patents
P.O. Box 1450
Alexandria, VA 22313-1450

Dear Sir/Madam:

In Response to the Examiner's Answer filed July 7, 2009, Appellant presents this Reply Brief. Appellant respectfully requests that this Reply Brief be considered by the Board of Patent Appeals and Interferences.

Applicant respectfully notes that the Examiner's Answer of July 7, 2009 replaced a previous Examiner's Answer (of October 28, 2008) vacated due to a typographical error, and indicated that Appellant's Reply Brief of December 12, 2008 has been entered and considered. Moreover, the second Examiner's Answer does not appear to include any new or modified arguments with respect to the previous Examiner's Answer. To avoid any potential confusion, Applicant submits this second Reply Brief, which restates the arguments of the previous Reply Brief.

STATUS OF CLAIMS

Claims 1-68 have been cancelled. Claims 69-92 are pending in the case. Claims 69-92 stand rejected under 35 U.S.C. § 102(b) and are the subject of this appeal. A copy of claims 69-92, incorporating entered amendments, as on appeal, is included in the Claims Appendix hereto.

GROUNDS OF REJECTION TO BE REVIEWED ON APPEAL

Claims 69-92 were rejected under 35 U.S.C. 102(b) as being anticipated by Kudukoli (US Pat. Pub. No. 2001/0024211 A1).

ARGUMENTS

The below is presented in response to the Examiner's Response to Argument in the Examiner's Answer. Appellant respectfully notes that the Examiner's Answer substantially repeats arguments made earlier in the prosecution, which have been addressed in the Appeal Brief. The Examiner's Answer includes further citations and arguments directed to Kudukoli. Appellant has presented below responses to any *new* citations and arguments in the Examiner's Answer. Thus, where no new discernable arguments were presented in the Examiner's Answer, Appellant has not added further responses, but relied on those presented in the Appeal Brief.

Ground of Rejection

Claims 69-92 were rejected under 35 U.S.C. 102(b) as being anticipated by Kudukoli (US Pat. Pub. No. 2001/0024211 A1). Appellant respectfully traverses this rejection for the following reasons. Different groups of claims are addressed under their respective subheadings.

Appellant respectfully notes that in the Examiner's Answer the Examiner cited numerous portions of Kudukoli, but provided no reasoning or explanation as to how these citations relate to the claim limitations.

Claims 69, 77

Claims 69 and 77 are separately patentable because the cited reference does not teach or suggest the limitations recited in these claims.

In the Examiner's Answer, the Examiner continues to argue that Kudukoli teaches **associating the determined graphical program code with the node, wherein, when the node in the graphical program executes, the determined graphical program code executes to provide the functionality in accordance with the selected function**, as recited in claim 69, citing the additional Figures 25A-C and paragraph [0022].

Cited Figures 25A-C illustrate a graphical program that includes various nodes for graphical program creation, specifically, a GPG (graphical program generation) program for creating the graphical program of Figures 23 and 24.

Cited paragraph [0022] reads:

> [0022] In other embodiments, the GPG program may include or be associated with a program or application that directly aids the user in creating a graphical program. For example, the GPG program may be included in a graphical programming development application. In this case the graphical programming development application may be operable to receive user input specifying desired functionality and the GPG program may automatically, i.e., programmatically, add a portion of graphical program code implementing the specified functionality to the user's program. The user input may be received, for example, via one or more "wizard" user interface dialogs enabling the user to specify various options. Such graphical program code generation wizards may greatly simplify the user's task of implementing various operations. As an example, it is often difficult for developers of instrumentation applications to properly implement code to analyze an acquired signal, due to the inherent complexity involved. By enabling the developer to easily specify the desired functionality through a high-level user interface, the GPG program can receive this information and automatically create graphical code to implement the signal analysis. Furthermore, since the graphical code is generated programmatically, the code may be optimized, resulting in an efficient program and a readable block diagram without unnecessary code. (*emphasis added*)

As discussed in the Appeal Brief, Appellant respectfully notes that:

1) the GPG program generates graphical program code for a *new* graphical program, not for itself. For example, the cited New VI Object Reference node in the GPG program is configured to create a new VI object (e.g., based on a VI object style input value), such as, for example, the cited waveform chart control and waveform chart user interface node, then when the GPG program is run, this New VI Object Reference node executes and creates a new VI object of the specified style in a *new graphical program*; and

2) the program code that creates the specified new VI object at runtime is inherently already part of the New VI Object Reference node, and, per 1), any graphical program code generated by the New VI Object Reference node belongs to the new graphical program.

Thus, in Kudukoli, there is no *associating* determined code with the New VI Object Reference node. Nowhere does Kudukoli mention or even hint at determining graphical program code for a node based on user-selection of displayed functions for the node, and *associating the determined graphical program with the node, where when the node executes, the determined graphical program code executes to perform the function.*

In fact, Kudukoli nowhere describes associating determined graphical program code with a node that is already displayed in a graphical program at all. The only associating Kudukoli discusses is between the GPG program and the received program information ([0037], [0147]), between the GPG program and a program or application that aids the user in creating the GPG program or a new graphical program, e.g., a development environment or application ([0022] and elsewhere), and between a new graphical program and a new programming environment ([0023] and elsewhere).

Thus, Appellant submits that even if Kudukoli's New VI Object Reference node code that generates a new VI object for new graphical program were considered to be Appellant's claimed "determined graphical program code", this code (of the New VI Object Reference node) inherently belongs to the New VI Object Reference node, and thus no *associating* of this code with the node is performed. Nowhere does Kudukoli describe associating determined graphical program code with the New VI Object Reference node based on selected functionality for the node. Thus, the Advisory Action's assertion that Kudokoli teaches associating code for generating the waveform chart (or random number generator or stop button) with the node is incorrect, since this code already belongs to the node.

The Examiner has improperly read this claimed feature into Kudokoli without presenting any evidence to support the feature. Nowhere does Kudukoli disclose this feature, nor does Kudokoli indicate anywhere that the code of the New VI Object Reference node that creates the graphical program object is itself graphical program code.

Similarly, as Appellant notes above, any new VI object created by the New VI Object Reference node *belongs to the new graphical program*, and its code is neither associated with the New VI Object Reference node, nor executed when the New VI Object Reference node executes; rather, the new VI object is created when the New VI Object Reference node is executed. Appellant submits that the Examiner appears to be

blurring the distinction between the inherent code of the New VI Object Reference node that creates new nodes/objects for the new graphical program, and the created nodes/objects themselves, which is improper and incorrect. Moreover, the Examiner appears to be ignoring the fact that in Kudukoli the New VI Object Reference node is in a first graphical program, while the generated new VI object is in a different (automatically generated) graphical program. This is in direct contrast with Appellant's invention as recited in claim 69, where there is only one graphical program that includes the original node, both before and after the claimed associating.

Additionally, in the Response to Arguments, the Examiner asserts that Kudukoli teaches that after the New VI Object Reference Node receives user input selecting a vi object class and a style/subclass, the New VI Object Reference Node "becomes/is replaced by a Vertical Pointer Slide node", citing [0213] and [0217]. This is *incorrect*. Applicant respectfully notes that cited paragraph [0213] clearly states that "The New VI Object Reference node creates a new VI object and outputs a reference to the new VI object."

As a careful reading of Kudukoli makes clear, the New VI Object Reference Node is in the *GPG program* (such as, for example, the GPG program of Figures 25A-C), while the generated node, such as the cited Vertical Pointer Slide node, which Appellant notes is generated by the New VI Object Reference Node when the GPG program executes, is in the *newly generated graphical program*, such as, for example, the generated graphical program of Figures 23 and 24, which was created by the GPG program of Figure 25.

Appellant further notes that if Kudukoli operated as the Examiner argues, the GPG of Figure 25 would, upon execution, modify itself, rather than generating the graphical program of Figures 23 and 24, which is clearly not the case. In other words, the Examiner appears to argue that the GPG program becomes the newly generated program, which is not the case. Rather, the GPG program executes to generate a *new* graphical program. This is why the GPG program is called a *graphical program generation program*.

Moreover, the Examiner cites Figure 4, arguing that blocks 208-210 explicitly teach "when the node in the graphical program executes, the determined graphical

program code executes to provide the functionality in accordance with the selected function", per Appellant's claim 69, lines 12-14. The Examiner then presents numerous figures and arguments directed to the notion that the New VI Object Reference Node is replaced by the object that it creates when executed. Again, this is *incorrect*. The Examiner repeatedly fails to distinguish between the code of the New VI Object Reference Node (determined by node inputs) that generates an object, and the generated object itself. Said another way, the Examiner fails to distinguish between the code that the New VI Object executes (which may or may not be graphical program code, and which is nowhere described as being generated in response to user input) to generate the specified object (which *is* the generated graphical program code) and the generated object/graphical program code. Appellant respectfully notes that the functionality of the New VI Object Reference node that generates the object is *not* the same as the functionality of the generated object, a distinction that the Examiner fails to make.

For example, the Examiner quotes [0278], directed to Figure 25, section 2, which describes using a New VI Object Reference node to create a waveform chart user interface control. Now, per Kudukoli, when the New VI Object Reference node executes, it generates the waveform control, and outputs a reference to it. Note that the New VI Object Reference node creates this waveform control in accordance with the inputs provided to the node, which determines, for example, the waveform control's position in a front panel of the newly generated graphical program when created. Note that the New VI Object Reference node is in the block diagram of the GPG program of Figure 25, whereas the newly created waveform control is in the newly created front panel of Figure 23, and has a corresponding waveform chart user interface node (618) in the newly generated block diagram of Figure 24.

Note, therefore, that the functionality of the New VI Object Reference node (determined by input to the node) is to generate the waveform chart/control, whereas the functionality of the generated waveform chart/control is to display waveform data, which is quite different.

The Examiner makes similar arguments regarding a "wait function node", which are similarly incorrect. Appellant respectfully submits that similar arguments apply to this case.

Thus, Kudukoli fails to teach or suggest all the features and limitations of claim 69, and so Appellant respectfully submits that the teachings of Kudukoli clearly do not meet the standard for anticipation which requires that the identical invention must be shown in as complete detail as is contained in the claims. *Richardson v. Suzuki Motor Co.*, 868 F.2d 1226, 1236, 9 USPQ2d 1913, 1920 (Fed. Cir. 1989). Anticipation requires the presence in a single prior art reference disclosure of each and every element of the claimed invention, arranged as in the claim. *Lindemann Maschinenfabrik GmbH v. American Hoist & Derrick Co.*, 730 F.2d 1452, 1457, 221 USPQ 481, 485 (Fed. Cir. 1984). Appellants' invention as recited in claim 69 is clearly not anticipated by Kudukoli, and so claim 69 is allowable.

Claim 79

Claim 79 is separately patentable because the cited reference does not teach or suggest **wherein said changing the first node icon to a second appearance comprises replacing the first node icon with a second node icon**.

In the Examiner's Answer, the Examiner further cites paragraph [0225], which reads:

> vi object class specifies a class to cast the object reference to. FIG. 21 illustrates how a user may choose an object class from a hierarchical menu. For example a reference to a vertical pointer slide object may be upcasted to a reference to a general slide object.

As may be seen, this citation is not germane to changing the first node icon to a second appearance comprises replacing the first node icon with a second node icon. Thus, Kudukoli fails to teach this feature of claim 79, and so claim 79 is patentably distinct and nonobvious, and thus allowable.

Regarding dependent claims 70-71 and 78-79, specifically, the limitations "wherein the node has a first node icon which is displayed in the graphical program, and wherein the first node icon has a first appearance, wherein the program instructions are further executable to perform: changing the first node icon to a second appearance based

on the second user input, wherein said changing the first node icon to a second appearance includes displaying an image corresponding to the selected function" (per claim 70), in the Response to Arguments, the Examiner asserts that the New Object VI Reference node changes its appearance to either a waveform chart UI node or a wait function node. This is incorrect.

As explained above, the New Object VI Reference node executes (in the GPG program) to generate these other objects, e.g., the waveform chart UI node or a wait function node, in a newly generated graphical program. Thus, the New Object VI Reference node does not change its appearance to look like these created objects.

Regarding dependent claims 72 and 80, specifically, the limitations "wherein, prior to said associating the determined graphical program code with the node, the node does not have any associated graphical program code" (per claim 72), in the Response to Arguments, the Examiner asserts that cited paragraphs [0212]-[0221] (and Figure 13) explicitly teach that "the New Object VI Reference node does not have any associated graphical program node before receiving first and second user input". This is incorrect.

The citations describe the New Object VI Reference node in some detail, but nowhere states or even hints that prior to receiving user input determining the object to generate, the New Object VI Reference node has no code implementing its functionality.

Claim 81

Claim 81 is separately patentable because the cited reference does not teach or suggest **wherein, prior to said associating the determined graphical program code with the node, the node has associated default graphical program code in accordance with a default function for the node, and wherein the default graphical program code implements a first functionality; and wherein said associating the determined graphical program code with the node comprises replacing the default graphical program code with the determined graphical program code**.

In the Examiner's Answer, the Examiner further cites paragraph [0130], which reads:

> [0130] In a typical case, the implementation of the graphical program code is determined mainly or entirely by the GPG program, although the received information may influence the manner in which the GPG program generates the code, or the GPG program may receive separate information influencing the code generation. For example, consider a GPG program operable to translate an existing graphical program to a new graphical program, e.g., in order to port the existing graphical program to a new programming environment. In one embodiment, the GPG program may be operable to generate the new graphical program in such a way as to match the existing graphical program as closely as possible in appearance. In other words, the new graphical program may be generated so that when the user sees the block diagram of the new graphical program, the block diagram appears substantially the same as the block diagram of the existing graphical program, e.g., in terms of the layout and connections among the block diagram nodes. In another embodiment, the GPG program may be operable to implement the new graphical program differently, e.g., by optimizing the code where possible. In this example, the functionality of the generated graphical program may be the same in either case, but the graphical program may be implemented in different ways.

As may be seen, this citation is not germane to a node having associated default graphical program code in accordance with a default function for the node, where the default graphical program code implements a first functionality, nor replacing the default graphical program code with the determined graphical program code.

Cited paragraph [0269] describes a VI refnum, specifically, a VI Server front panel refnum control being placed in a front panel (a GUI for a graphical program), and configured to be some specified type of refnum, after which the refnum control us the appearance of the specified type of refnum. Appellant notes that a refnum control is not a graphical program node for use in a graphical program, but rather is a type of control for use in a front panel (GUI) that provides a reference to an appropriate entity, e.g., to an application, VI, etc. Thus, this citation is also not germane to these claimed limitations.

Cited paragraph [0230] is directed to an input of a Downcast Reference node, specifically, a "vi object class" input that specifies a class to downcast the object reference to, which also has nothing whatsoever to do with a node having default graphical program code prior to associating determined graphical program code with the node.

Thus, Kudukoli fails to teach this feature of claim 81, and so claim 81 is patentably distinct and nonobvious, and thus allowable.

In the Response to Arguments, the Examiner asserts that Kudukoli explicitly teaches "'upcasting' and/or 'downcasting' in paragraphs [0222]-[0230], which may cast the type of a general slide object (i.e., a superclass with a default graphical program code) to the type of a vertical pointer slide object (i.e., replacing the superclass with a default graphical program code with the graphical program code of the vertical pointer slide object". This is not only incorrect, but technically nonsensical. Applicant respectfully notes that this node, when executed, casts a reference of a an object (such as an object created by the New VI Object Reference node) to the type of a subclass. As one of skill in the programming arts would readily understand, downcasting a reference changes (usually temporarily) the data type of the reference, but does not replace "the superclass with a default graphical program code with the graphical program code of the vertical pointer slide object", as argued by the Examiner.

Thus, Kudukoli fails to teach this feature of claim 81, and so claim 81 is patentably distinct and nonobvious, and thus allowable.

Claim 82

Claim 82 is separately patentable because the cited reference does not teach or suggest **wherein said receiving first user input comprises receiving the first user input to the node; and wherein said receiving second user input comprises receiving the second user input to the node**.

In the Examiner's Answer, the Examiner further cites paragraph [0022], which reads:

> [0022] In other embodiments, the GPG program may include or be associated with a program or application that directly aids the user in creating a graphical program. For example, the GPG program may be included in a graphical programming development application. In this case the graphical programming development application may be operable to receive user input specifying desired functionality and the GPG program may automatically, i.e., programmatically, add a portion of graphical

> program code implementing the specified functionality to the user's program. The user input may be received, for example, via one or more "wizard" user interface dialogs enabling the user to specify various options. Such graphical program code generation wizards may greatly simplify the user's task of implementing various operations. As an example, it is often difficult for developers of instrumentation applications to properly implement code to analyze an acquired signal, due to the inherent complexity involved. By enabling the developer to easily specify the desired functionality through a high-level user interface, the GPG program can receive this information and automatically create graphical code to implement the signal analysis. Furthermore, since the graphical code is generated programmatically, the code may be optimized, resulting in an efficient program and a readable block diagram without unnecessary code.

As may be seen, this citation discusses receiving user input via "wizard" user interface dialogs, not to the node, and so this citation is not germane to these claimed features.

Cited paragraph [0034] reads:

> [0034] The graphical program may be modified in any of various ways. As described above, the graphical program may have a block diagram that includes graphical source code, such as interconnected nodes (e.g., function nodes), programmatic structures (e.g., loops), etc., and the graphical source code of the block diagram may be modified. For example, graphical source code may be added or removed, interconnections among block diagram nodes may be changed, etc. Also, the graphical program may have a user interface portion that may be modified. The GPG program may be operable to programmatically perform a plurality of modifications or types of modifications to graphical programs, depending on the received program information.

Clearly, this citation also has nothing to do with receiving user input to a node invoking display of a plurality of functions for the node or selecting a function from the plurality of functions.

In the Response to Arguments, the Examiner argues that the *object* class and type specification via hierarchical menus of Figures 21 and 22 (and paragraphs [0215-0217] and [0278]) teaches Applicant's claimed user selection of *functions*, specifically, user selection of a function class and a function of that class. Applicant respectfully submits

that an *object* is not a *function*. Thus, Kudukoli fails to anticipate this claimed feature per the strict requirements of an anticipatory rejection under section 102(b).

Thus, Kudukoli fails to teach this feature of claim 82, and so claim 82 is patentably distinct and nonobvious, and thus allowable.

Claim 83

Claim 83 is separately patentable because the cited reference does not teach or suggest **wherein said displaying the plurality of functions for the node in response to the first user input comprises: displaying a plurality of function classes for the node; and in response to user input selecting a function class, displaying the plurality of functions, wherein the plurality of functions are in the selected function class**.

In the Examiner's Answer, the Examiner further cites paragraphs [0222]-[0230], and Figure 25A, section 2.

Similar to above (wrt claim 82), these citations are directed to user (or programmatic) specification of *object* classes and *object* types, not *function* classes and *functions*, and so fail to meet the strict requirements of an anticipatory rejection under section 102(b).

Thus, Kudukoli fails to teach this feature of claim 83, and so claim 83 is patentably distinct and nonobvious, and thus allowable.

Claim 84

Claim 83 is separately patentable because the cited reference does not teach or suggest **wherein the node is a data acquisition (DAQ) node; wherein the plurality of functions for the node comprise a plurality of DAQ functions; wherein, prior to said associating, the DAQ node comprises one of:**

a generic read node;

a generic write node;

a generic channel creation node;

a generic timing node; or

a generic triggering node; and

wherein, after said associating, the DAQ node comprises one of:

a specific read node in accordance with the selected function;

a specific write node in accordance with the selected function;

a specific channel creation node in accordance with the selected function;

a specific timing node in accordance with the selected function; or

a specific triggering node in accordance with the selected function.

In the Examiner's Answer, the Examiner further cites Figure 23, Figure 25C (section 8), and paragraphs [0275] and [0284], arguing that these citations disclose that after an association process, a DAQ node, specifically, a wait node, is replaced with a specific timing node.

This is not the manner in which Kudukoli operates. Rather, these citations clearly teach that a New VI Object Reference node in the GPG program is used to create "wait function node" in a newly generated graphical program. The input cited by the Examiner refers to input to the New VI Object Reference node that specifies the creation of the wait function node. There is no "association" process in Kudukoki that changes a DAQ node from a generic read/write/channel creation/timing/triggering node to a specific read/write/channel creation/timing/triggering node, as claimed.

Thus, Kudukoli fails to teach this feature of claim 84, and so claim 84 is patentably distinct and nonobvious, and thus allowable.

Claim 85

Claim 85 is separately patentable because the cited reference does not teach or suggest **determining a second node based on the selected function, wherein the second node comprises a graphical representation of an implementation of the selected function, and wherein the second node comprises graphical program code executable to provide functionality in accordance with the selected function**.

In the Examiner's Answer, the Examiner further cites paragraphs [0034], which reads:

Cited paragraph [0034] reads:

> [0034] The graphical program may be modified in any of various ways. As described above, the graphical program may have a block diagram that includes graphical source code, such as interconnected nodes (e.g., function nodes), programmatic structures (e.g., loops), etc., and the graphical source code of the block diagram may be modified. For example, graphical source code may be added or removed, interconnections among block diagram nodes may be changed, etc. Also, the graphical program may have a user interface portion that may be modified. The GPG program may be operable to programmatically perform a plurality of modifications or types of modifications to graphical programs, depending on the received program information.

As may be seen, this citation does not disclose or even hint at determining a second node based on the selected function, wherein the second node comprises a graphical representation of an implementation of the selected function.

Moreover, regarding the limitation "wherein the second node comprises graphical program code executable to provide functionality in accordance with the selected function", newly cited paragraphs [0136]-[0140] read:

> [0136] FIG. 6--Programmatically Modifying a Graphical Program
>
> [0137] Another embodiment of the invention comprises a system and method for programmatically modifying a graphical program. FIG. 6 is a flowchart illustrating one embodiment of a method for programmatically modifying a graphical program.
>
> [0138] In step 300, a GPG program such as described above may be executed.
>
> [0139] In step 302, the GPG program may receive initial program information specifying functionality of a graphical program (or graphical program portion), similarly as described above.
>
> [0140] In step 304, the GPG program may programmatically generate a graphical program (or graphical program portion) to implement the specified functionality, similarly as described above.

The citation describes a GPG program generating a graphical program (or portion) in response to receiving "initial program information specifying functionality of a graphical program (or graphical program portion)", but makes no mention of a second

node (that was determined based on a user-selected function for a first node) replacing the first node in a graphical program at all, and so cannot teach the second node including graphical program code executable to provide functionality in accordance with the selected function.

In the Response to Arguments, the Examiner further cites paragraphs [0212]-[0217], which are directed to the New VI Object Reference node. Applicant respectfully notes that per Kudukoli, the New VI Object Reference node creates an object (e.g., a node) in a *new* graphical program based on input specifying the class and type of the object, and specifically does *not* teach determining a replacement node to perform a specified function in the graphical program that includes the original node, which in Kudukoli would be the GPG program. In other words, in Kudukoli, no nodes in the GPG program are replaced, and no nodes are determined to replace any nodes in either the GPG program or the new graphical program. Thus, Kudukoli fails to teach or suggest this feature of claim 85.

The Examiner further argues that Figures 25A and 25C teach that the New VI Object Reference node is replaced with the waveform chart UI node or the wait function node. As explained above at length, this is not the case. The New VI Object Reference node, which is in the GPG program, executes to create the waveform chart UI node or the wait function node in a *new* graphical program. This does *not* alter the GPG program in any way.

Cited Figure 22 illustrates user specification of an object to be created in the new graphical program by the New VI Object Reference Node (in the GPG program). As explained above with respect to claim 69, Kudukoli nowhere teaches or suggests, or even hints at, determining a second node for replacing a first node in a graphical program, as claimed.

Thus, Kudukoli fails to teach this feature of claim 85.

Nor does the cited art disclose **replacing the node in the graphical program with the second node, wherein, when the second node in the graphical program**

executes, the graphical program code of the second node executes to provide the functionality in accordance with the selected function, as also recited in claim 85.

In the Response to Arguments, the Examiner argues that "the second node (either the waveform chart UI node in FIG. 25A, Section 2 or the wait function node in FIG. 25C, Section 8) has the first node icon of the first node (the node icon of New VI Object Reference Node) and the newly created graphical program node (used/connect/associated with either the wafeform chart UI node or the wait function node)", citing Figure 13.

This is incorrect. The newly created waveform chart UI node or wait function node are not in the GPG program of Figure 25, but rather are generated by respective New VI Object Reference nodes configured to generate these nodes for and in the new graphical program. These created nodes never have the icon of the New VI Object Reference Node, contrary to the Examiner's assertion. Nor does the New VI Object Reference Node ever have the icons or graphical program code of the generated chart UI node or wait function node.

The Examiner's annotations of these figures mistakenly asserts that the New VI Object Reference Node shown in these figures is the chart UI node or the wait function node. This is incorrect. These generated nodes are *only* in the newly created graphical program node–they are *never* placed in the GPG program. Applicant respectfully submits that the Examiner may be referring to the specified inputs shown with respect to the New VI Object Reference nodes displayed, specifically, the object type specification shown being input to the New VI Object Reference nodes specifying creation of the chart UI node or wait function node, respectively. These are not node icons.

Nowhere do the citations, nor Kudukoli in general, teach or suggest replacing any node in the GPG program with the created nodes. Thus, Kudukoli fails to teach this feature of claim 85.

Thus, Kudukoli fails to teach all the features and limitations of claim 85, and so claim 85 is patentably distinct and nonobvious, and thus allowable.

Claims 87 and 91

Claim 87 is separately patentable because the cited reference does not teach or suggest **wherein the node and/or the second node is one or more of: polymorphic; function switchable; or function class switchable**.

In the Response to Arguments, the Examiner further argues that Kudukoli "explicitly teaches a New VI Object Reference Node (FIG. 13) may be replaced by either a waveform chart UI node (FIG. 25A, Section 2) or a wait function node (FIG. 25C, Section 8), which clearly the NEW VI Object Reference Node is polymorphic [*sic*]".

As explained above, the New VI Object Reference node executes in the GPG program to create the waveform chart UI node or wait function node in the new graphical program. In other words, the newly created waveform chart UI node or wait function node are not in the GPG program of Figure 25, but rather are generated by respective New VI Object Reference nodes in the GPG program that have been configured to generate these nodes for and in the new graphical program.

Thus, Kudukoli fails to teach this feature of claim 87, and so claim 87 is patentably distinct and nonobvious, and thus allowable.

CONCLUSION

For at least the foregoing reasons, it is respectfully submitted that the Examiner's rejection of claims 69-92 was erroneous, and reversal of the decision is respectfully requested.

The Commissioner is authorized to charge any fees that may be due to Meyertons, Hood, Kivlin, Kowert, & Goetzel, P.C. Deposit Account No. 501505/5150-80201/JCH.

Respectfully submitted,

/Jeffrey C. Hood/

Jeffrey C. Hood, Reg. #35,198
ATTORNEY FOR APPLICANT(S)

Meyertons Hood Kivlin Kowert & Goetzel, P.C.
P.O. Box 398
Austin, TX 78767-0398
Phone: (512) 853-8800

Date: 2009-08-24 JCH/MSW